案例 8-多边形有效边表填充算法

文档编写：霍波魏

校稿/修订：孔令德

时间 2019~2020

联系方式：QQ997796978

说明：本套案例由孔令德开发，原版本为 Visual C++6.0，配套于孔令德的著作《计算机图形学-基于 MFC 三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于 Windows 10 操作系统，使用 Microsoft visual studio 2017 平台的 MFC（英文版）开发。

一、知识点

1. 有效边表

多边形内与当前扫描线相交的边称为有效边（AE），将有效边按照与扫描线交点的 X 坐标递增的顺序存放在一个链表中，成为有效边表（AET）。

2. 多边形填充

多边形填充是通过顶点颜色的线性插值，求出各个顶点之间的每个像素点的颜色并将其填充。

二、案例描述

本案例将有效边表算法和多边形填充算法相结合，绘制出一个正六边形，并通过颜色的线性插值在六个颜色各异的顶点之间实现颜色的平滑着色。

三、实现步骤

1. 添加 CRGB 类、CP2 类和 CP2i 类。
2. 添加有效边表 AET 类和桶表类 Bucket 类。
3. 添加六边形 CHexagon 类，给定了点表，绘制了多边形边界并填充了多边形。
4. 添加填充类 Fill 类并在其中定义桶表、边表函数，边表排序函数、合并边表函数、多边形填充函数、线性插值函数、清理内存函数以及删除边表函数。
5. 在 CTestView 类中定义双缓冲函数、绘制多边形函数。

四、主要算法

1.CHexagon 类

```
public:
    CHexagon();
    virtual~CHexagon();
    void ReadPoint();//读入顶点表
    void DrawPolygon(CDC *pDC);//绘制多边形
    void FillPolygon(CDC *pDC);//填充多边形
public:
    CP2 P[7];//浮点点表
    CPi2 PO[7];//y整数点表
    BYTE bRed, bGreen, bBlue;//颜色分量
    BOOL bFill;//填充标志;
void CHexagon::ReadPoint()//点表
{
    int r = 300;
    double w = PI / 180;
    P[0] = CP2(r, 0, CRGB(1.0, 0.0, 0.0));
    P[1] = CP2(r*cos(w * 60), r*sin(w * 60), CRGB(1.0, 1.0, 0.0));
    P[2] = CP2(-r * cos(w * 60), r*sin(w * 60), CRGB(0.0, 1.0, 0.0));
    P[3] = CP2(-r, 0, CRGB(0.0, 1.0, 1.0));
    P[4] = CP2(-r * cos(w * 60), -r * sin(w * 60), CRGB(0.0, 0.0, 1.0));
    P[5] = CP2(r*cos(w * 60), -r * sin(w * 60), CRGB(1.0, 0.0, 1.0));
}
void CHexagon::DrawPolygon(CDC *pDC)//绘制多边形边界
{
    CLine *line = new CLine;
    CP2 t;
    for (int i = 0; i < 6; i++)//绘制多边形
    {
        if (i == 0)
        {
            line->MoveTo(pDC, P[i]);
            t = P[i];
        }
        else
        {
            line->LineTo(pDC, P[i]);
        }
    }
    line->LineTo(pDC, t);//闭合多边形
    delete line;
}
void CHexagon::FillPolygon(CDC *pDC)//填充多边形
{
```

```
    for (int i = 0; i < 6; i++)
    {
        P0[i].x = P[i].x;
        P0[i].y = Round(P[i].y);
        P0[i].c = P[i].c;
    }
    CFill *fill = new CFill;      //动态分配内存
    fill->SetPoint(P0, 6);       //初始化Fill对象
    fill->CreateBucket();        //建立桶表
    fill->CreateEdge();          //建立边表
    fill->Gouraud(pDC);          //填充多边形
    delete fill;                 //撤销内存
}
```

2.CFill 类

```
public:
    CFill(void);
    virtual ~CFill(void);
    void SetPoint(CPi2 *p, int);//类的初始化
    void CreateBucket();//创建桶
    void CreateEdge();//边表
    void AddEt(CAET *);//合并ET表
    void EtOrder();//ET表排序
    void Gouraud(CDC *);//填充多边形
    CRGB Interpolation(double, double, double, CRGB, CRGB);//线性插值
    void ClearMemory();//清理内存
    void DeleteAETChain(CAET* pAET);//删除边表
public:
    int     PNum;//顶点个数
    CPi2    *P;//顶点坐标动态数组
    CAET    *pHeadE, *pCurrentE, *pEdge; //有效边表结点指针
    CBucket *pHeadB, *pCurrentB;        //桶表结点指针
CFill::CFill(void)
{
    PNum = 0;
    P = NULL;
    pEdge = NULL;
    pHeadB = NULL;
    pHeadE = NULL;
}
CFill::~CFill(void)
{
    if (P != NULL)
    {
        delete[] P;
```

```
            P = NULL;
        }
        ClearMemory();
}
void CFill::SetPoint(CPi2 *p, int m)
{
        P = new CPi2[m];//创建一维动态数组
        for (int i = 0; i < m; i++)
        {
            P[i] = p[i];
        }
        PNum = m;
}
void CFill::CreateBucket()//创建桶表
{
        int yMin, yMax;
        yMin = yMax = P[0].y;
        for (int i = 0; i < PNum; i++)//查找多边形所覆盖的最小和最大扫描线
        {
            if (P[i].y < yMin)
            {
                yMin = P[i].y;//扫描线的最小值
            }
            if (P[i].y > yMax)
            {
                yMax = P[i].y;//扫描线的最大值
            }
        }
        for (int y = yMin; y <= yMax; y++)
        {
            if (yMin == y)//如果是扫描线的最小值
            {
                pHeadB = new CBucket;//建立桶的头结点
                pCurrentB = pHeadB;//pCurrentB为CBucket当前结点指针
                pCurrentB->ScanLine = yMin;
                pCurrentB->pET = NULL;//没有链接边表
                pCurrentB->pNext = NULL;
            }
            else//其他扫描线
            {
                pCurrentB->pNext = new CBucket;//建立桶的其他结点
                pCurrentB = pCurrentB->pNext;
                pCurrentB->ScanLine = y;
                pCurrentB->pET = NULL;
```

```
            pCurrentB->pNext = NULL;
        }
    }
}
void CFill::CreateEdge()//创建边表
{
    for (int i = 0; i < PNum; i++)
    {
        pCurrentB = pHeadB;
        int j = (i + 1) % PNum;//边的第2个顶点，P[i]和P[j]点对构成边
        if (P[i].y < P[j].y)//边的终点比起点高
        {
            pEdge = new CAET;
            pEdge->x = P[i].x;//计算ET表的值
            pEdge->yMax = P[j].y;
            pEdge->k = (P[j].x - P[i].x) / (P[j].y - P[i].y);//代表1/k
            pEdge->ps = P[i];//绑定顶点和颜色
            pEdge->pe = P[j];
            pEdge->pNext = NULL;
            while (pCurrentB->ScanLine != P[i].y)//在桶内寻找当前边的yMin
            {
                pCurrentB = pCurrentB->pNext;//移到yMin所在的桶结点
            }
        }
        if (P[j].y < P[i].y)//边的终点比起点低
        {
            pEdge = new CAET;
            pEdge->x = P[j].x;
            pEdge->yMax = P[i].y;
            pEdge->k = (P[i].x - P[j].x) / (P[i].y - P[j].y);
            pEdge->ps = P[i];
            pEdge->pe = P[j];
            pEdge->pNext = NULL;
            while (pCurrentB->ScanLine != P[j].y)
            {
                pCurrentB = pCurrentB->pNext;
            }
        }
        if (P[i].y != P[j].y)
        {
            pCurrentE = pCurrentB->pET;
            if (pCurrentE == NULL)
            {
                pCurrentE = pEdge;
```

```
                    pCurrentB->pET = pCurrentE;
                }
                else
                {
                    while (pCurrentE->pNext != NULL)
                    {
                        pCurrentE = pCurrentE->pNext;
                    }
                    pCurrentE->pNext = pEdge;
                }
            }
        }
}
void CFill::Gouraud(CDC *pDC)//填充多边形
{
    CAET *pT1 = NULL, *pT2 = NULL;
    pHeadE = NULL;
    for (pCurrentB = pHeadB; pCurrentB != NULL; pCurrentB = pCurrentB->pNext)
    {
        for (pCurrentE = pCurrentB->pET; pCurrentE != NULL; pCurrentE =
pCurrentE->pNext)
        {
            pEdge = new CAET;
            pEdge->x = pCurrentE->x;
            pEdge->yMax = pCurrentE->yMax;
            pEdge->k = pCurrentE->k;
            pEdge->ps = pCurrentE->ps;
            pEdge->pe = pCurrentE->pe;
            pEdge->pNext = NULL;
            AddEt(pEdge);
        }
        EtOrder();
        pT1 = pHeadE;
        if (pT1 == NULL)
        {
            return;
        }
        while (pCurrentB->ScanLine >= pT1->yMax)//下闭上开
        {
            CAET * pAETTEmp = pT1;
            pT1 = pT1->pNext;
            delete pAETTEmp;
            pHeadE = pT1;
            if (pHeadE == NULL)
```

```
                return;
            }
            if (pT1->pNext != NULL)
            {
                pT2 = pT1;
                pT1 = pT2->pNext;
            }
            while (pT1 != NULL)
            {
                if (pCurrentB->ScanLine >= pT1->yMax)//下闭上开
                {
                    CAET* pAETTemp = pT1;
                    pT2->pNext = pT1->pNext;
                    pT1 = pT2->pNext;
                    delete pAETTemp;
                }
                else
                {
                    pT2 = pT1;
                    pT1 = pT2->pNext;
                }
            }
            CRGB Ca, Cb, Cf;//Ca、Cb代表边上任意点的颜色，Cf代表面上任意点的颜色
            Ca = Interpolation(pCurrentB->ScanLine, pHeadE->ps.y, pHeadE->pe.y,
pHeadE->ps.c, pHeadE->pe.c);
            Cb = Interpolation(pCurrentB->ScanLine, pHeadE->pNext->ps.y,
pHeadE->pNext->pe.y, pHeadE->pNext->ps.c, pHeadE->pNext->pe.c);
            BOOL Flag = FALSE;
            double xb, xe;//扫描线和有效边相交区间的起点和终点坐标
            for (pT1 = pHeadE; pT1 != NULL; pT1 = pT1->pNext)
            {
                if (Flag == FALSE)
                {
                    xb = pT1->x;
                    Flag = TRUE;
                }
                else
                {
                    xe = pT1->x;
                    for (double x = xb; x < xe; x++)//左闭右开
                    {
                        Cf = Interpolation(x, xb, xe, Ca, Cb);
                        pDC->SetPixel(ROUND(x), pCurrentB->ScanLine, RGB(Cf.red * 255,
Cf.green * 255, Cf.blue * 255));
```

```
                }
                Flag = FALSE;
            }
        }
        for (pT1 = pHeadE; pT1 != NULL; pT1 = pT1->pNext)//边的连续性
        {
            pT1->x = pT1->x + pT1->k;
        }
    }
}
void CFill::AddEt(CAET *pNewEdge)//合并ET表
{
    CAET *pCE = pHeadE;
    if (pCE == NULL)
    {
        pHeadE = pNewEdge;
        pCE = pHeadE;
    }
    else
    {
        while (pCE->pNext != NULL)
        {
            pCE = pCE->pNext;
        }
        pCE->pNext = pNewEdge;
    }
}
void CFill::EtOrder()//边表的冒泡排序算法
{
    CAET *pT1 = NULL, *pT2 = NULL;
    int Count = 1;
    pT1 = pHeadE;
    if (NULL == pT1)
    {
        return;
    }
    if (NULL == pT1->pNext)//如果该ET表没有再连ET表
    {
        return;//桶结点只有一条边，不需要排序
    }
    while (NULL != pT1->pNext)//统计结点的个数
    {
        Count++;
        pT1 = pT1->pNext;
```

```
}
for (int i = 1; i < Count; i++)//冒泡排序
{
    pT1 = pHeadE;
    if (pT1->x > pT1->pNext->x)//按x由小到大排序
    {
        pT2 = pT1->pNext;
        pT1->pNext = pT1->pNext->pNext;
        pT2->pNext = pT1;
        pHeadE = pT2;
    }
    else
    {
        if (pT1->x == pT1->pNext->x)
        {
            if (pT1->k > pT1->pNext->k)//按斜率由小到大排序
            {
                pT2 = pT1->pNext;
                pT1->pNext = pT1->pNext->pNext;
                pT2->pNext = pT1;
                pHeadE = pT2;
            }
        }
    }
    pT1 = pHeadE;
    while (pT1->pNext->pNext != NULL)
    {
        pT2 = pT1;
        pT1 = pT1->pNext;
        if (pT1->x > pT1->pNext->x)//按x由小到大排序
        {
            pT2->pNext = pT1->pNext;
            pT1->pNext = pT1->pNext->pNext;
            pT2->pNext->pNext = pT1;
            pT1 = pT2->pNext;
        }
        else
        {
            if (pT1->x == pT1->pNext->x)
            {
                if (pT1->k > pT1->pNext->k)//按斜率由小到大排序
                {
                    pT2->pNext = pT1->pNext;
                    pT1->pNext = pT1->pNext->pNext;
```

```
                        pT2->pNext->pNext = pT1;
                        pT1 = pT2->pNext;
                    }
                }
            }
        }
    }
}
CRGB CFill::Interpolation(double t, double t1, double t2, CRGB c1, CRGB c2)//线性插值
{
    CRGB c;
    c = (t - t2) / (t1 - t2)*c1 + (t - t1) / (t2 - t1)*c2;
    return c;
}
void CFill::ClearMemory()//安全删除所有桶和桶上面的边
{
    DeleteAETChain(pHeadE);
    CBucket *pBucket = pHeadB;
    while (pBucket != NULL)// 针对每一个桶
    {
        CBucket * pBucketTemp = pBucket->pNext;
        DeleteAETChain(pBucket->pET);
        delete pBucket;
        pBucket = pBucketTemp;
    }
    pHeadB = NULL;
    pHeadE = NULL;
}

void CFill::DeleteAETChain(CAET* pAET)
{
    while (pAET != NULL)
    {
        CAET* pAETTemp = pAET->pNext;
        delete pAET;
        pAET = pAETTemp;
    }
}
```

3.CTestView 类

```
public:
    void DrawGraph();//绘制图形
protected:
    CHexagon hex;
```

```
void CTestView::DrawGraph()//绘制图形
{
    CRect rect;                                          //定义客户区
    GetClientRect(&rect);                                //获得客户区的大小
    CDC *pDC = GetDC();                                    //定义设备上下文指针
    pDC->SetMapMode(MM_ANISOTROPIC);                     //自定义坐标系
    pDC->SetWindowExt(rect.Width(), rect.Height());       //设置窗口比例
    pDC->SetViewportExt(rect.Width(), -rect.Height());   //设置视区比例，且x轴水平向右，y轴垂直向上
    pDC->SetViewportOrg(rect.Width() / 2, rect.Height() / 2);//设置客户区中心为坐标系原点
    rect.OffsetRect(-rect.Width() / 2, -rect.Height() / 2);  //矩形与客户区重合
    hex.DrawPolygon(pDC);//绘制多边形
    hex.FillPolygon(pDC);//填充多边形
    ReleaseDC(pDC);//释放DC
}
```

五、实现效果

多边形有效边表填充算法效果如图 8-1 所示。

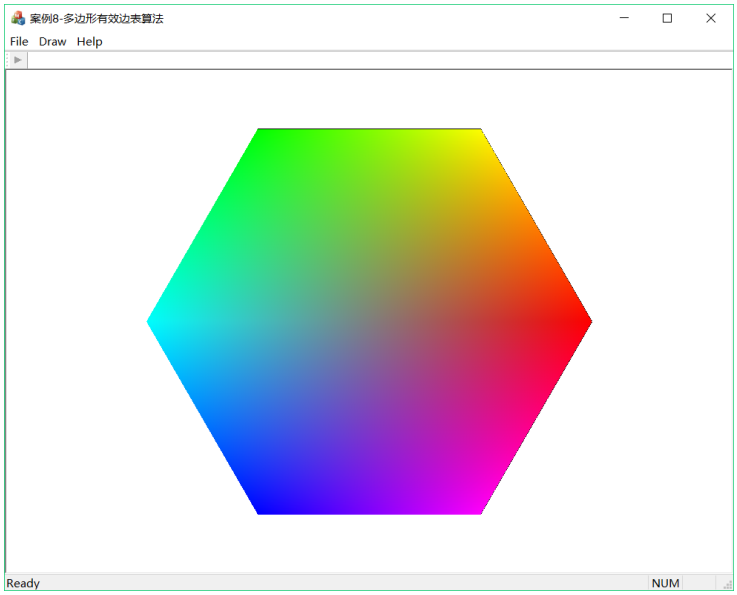

图 8-1 多边形有效边表填充算法效果图

六、遇到的问题及解决方案

有效边表的扫描线坐标：假设当前坐标为（X_i,Y_i）,那么下一个扫描线的交点为（X_{i+1},Y_{i+1}）,因为扫描线是沿着 Y 方向移动，所以 Y 的坐标采用加 1 操作，而 Y+1 的同时 X 坐标不一定加 1，已知斜率为有效边斜率为 k，所以 X 方向增加的距离为 1/k，即：

$$X_{i+1} = X_i + 1/k$$

$$Y_{i+1} = Y_i + 1$$

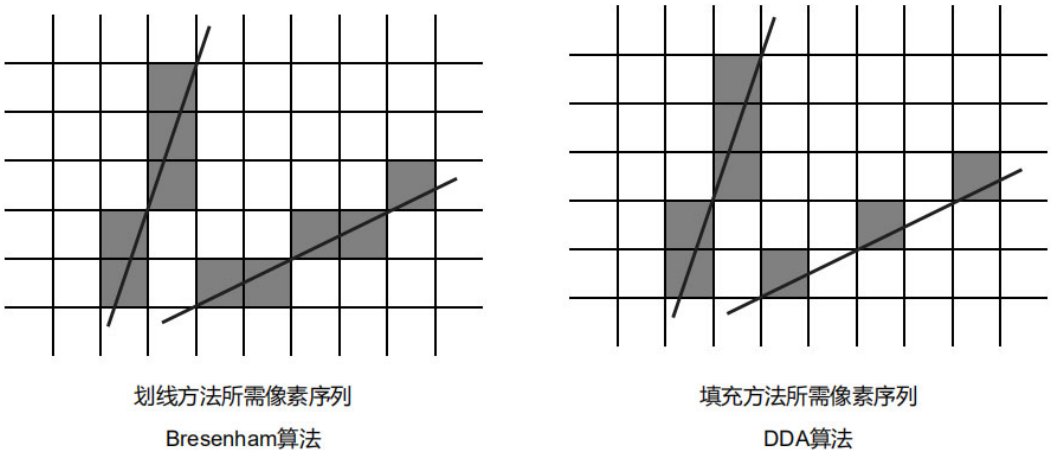

图 8-2 边的光栅化示意图

七、案例心得

多边形填充时边界的颜色问题是至关重要的，有效边表填充时需要注意边界像素的一般处理原则为“左闭右开，上闭下开”，利用这一原则，可以解决边界点、顶点的颜色问题。由于颜色填充的是有效边表中的像素点，所以在扫描线扫描的过程中，也需要遵循“左闭右开，上闭下开”这一原则。